

PDP-1 COMPUTER  
ELECTRICAL ENGINEERING DEPARTMENT  
M.I.T.  
CAMBRIDGE, MASSACHUSETTS 02139

PDP-29-1

INTRODUCTION TO PDP-1 TS SYSTEM PROGRAMS

September 21, 1966

## INTRODUCTION TO THE PDP-1 TIME SHARING SYSTEM

### GENERAL INFORMATION

This memo was written to give the new user a brief introduction to the PDP-1 time sharing system programs. The material presented is sufficient information needed to operate in the TS system; however, only the essential features of the system programs are described. Other useful and time saving features are described in the respective program memo.

The computer has three 4096-word core memories, one is used for executive control of the system and the other two are available to the users. In addition to these three 4096-word core memory, an extra amount of memory in the form of a magnetic drum is available. The storage space on the drum is broken up into a number of "fields", each of which has the capacity to store 4096 18-bit words. This drum memory (besides being used for the time-sharing operation) can be used to store programs and data when there is not enough room in core. It is also used as a permanent storage space for "utility" programs which can be used by you.

To execute a program that you have written for the PDP-1, you must do the following things:

- 1) Tell the computer, in some way, the sequence of machine-language instructions that make up your program.
- 2) Have the computer translate or assemble these instructions into a sequence of binary words.
- 3) Place this binary program into core memory and have it executed, and
- 4) check your answers to see if your program is working correctly.

The permanently stored utility programs will be of help to you in accomplishing the above tasks. One of the programs, EXPENSIVE TYPEWRITER, assists you in reading in your machine-language program, producing a binary program. DDT is an all-purpose program that will see to it that your program is executed. It also allows you to inspect and test your binary program for proper operation.

Each of the three utility programs mentioned above is stored (in binary form) on a different field of the magnetic drum. Of course, in order to use a program you must first place it in core memory. This is done by typing the commands shown in Figure 1.

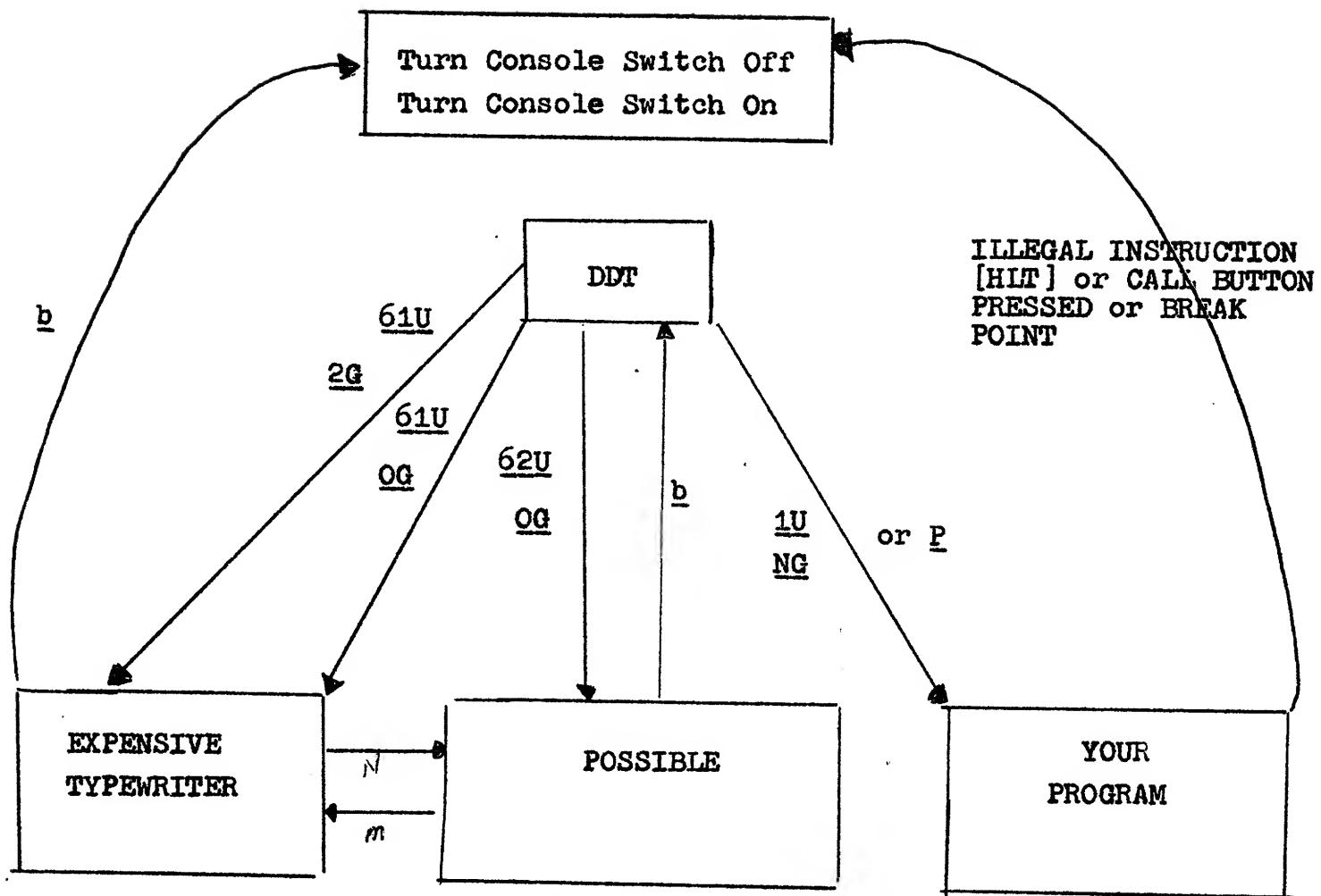


Figure 1.

For example, turning the console switch off, then turning it on, will bring the program DDT from drum to core. Anything typed by you at this point will be interpreted as a command to DDT. Typing 61U\* then OG will cause EXPENSIVE TYPEWRITER to be brought into core. Anything typed at this point will be interpreted as a command to EXPENSIVE TYPEWRITER.

The features of each of the programs will now be described.

---

\* The letters typed by you will be underlined in this paper. Those typed by the computer will not be underlined. The symbol m is a carriage return

EXPENSIVE TYPEWRITER:

This program is used for the reading in or typing in of your machine language program. It is also valuable in modifying your machine language program--with it you can do such things as changing lines, inserting lines, deleting lines, etc. This program has a buffer storage area where a copy of your machine language program is kept. When you enter EXPENSIVE TYPEWRITER by typing 61U then 0G to DDT, this buffer area is cleared. Entering E.T. by typing 61U then 2G will not clear this buffer.

E.T. has two operating "modes", control mode and text mode. In the control mode, characters typed in are commands to EXPENSIVE TYPEWRITER; in the text mode, typed input is copied directly into the buffer. The only exception to this is the character "backspace" which in the text mode has the effect of cancelling the previous character. To go from text mode to control mode, type a backspace immediately following a carriage return or after deleting all characters from a line.

Certain commands in the control mode put E.T. into the text mode. In the control mode, the typewriter prints in red, and in the text mode it prints in black. A lower case command is terminated by typing in a carriage return or tab; at any time before the terminator is typed the command can be cancelled by typing the character middle dot (•).

Commands to EXPENSIVE TYPEWRITER:

In the examples below, the letter N is used to signify the number of a line. (A carriage return denotes the end of a line.) For example, the title line of your program will be line one, etc. Slash (/) or period (.) may be used for the N to signify the last or current line number, respectively.

I. Input:

A: append. Enter the text mode; add the following typed text onto the end of the previous contents of the buffer; the buffer already has a stop code at the end.

Ni: insert. Enter the text mode; insert the following typed text immediately before line N. That is, the first line of the inserted text becomes the new line N. The text is squeezed in between the old lines N-1 and N; no material is lost.

Nc: change. Enter the text mode. Line N is entirely deleted and the following typed text is inserted in its place; any number of lines may be inserted.

r: read. The tape in the reader is read in and appended to the end of the buffer.

II. Deletion of Information in the Buffer:

Nd: delete. Line N is deleted.

K: kill. The entire buffer is cleared.

III. Printout:

W: write. The buffer is printed out (in black).

N1: line. Line N is printed out (in red).

backspace: prints out the next line

↑: prints out the previous line.

IV. Punching Tape

P: punch. This punches the buffer out on paper tape.

V. Transfer

N: transfer control to POSSIBLE

b: transfer control to DDT

VI. Search

S<sup>—</sup>: limited.

S<sup>∞</sup>: unlimited. Search the current page (limited) or the entire buffer (unlimited) for the first occurrence of the string anywhere in the text buffer. If none is found, ET types "missing". Otherwise the search command is replaced by the line number where the string was found.

Typical Operating Procedure:

Probably the first time you use the computer you will enter E.T. by typing 61U then 0G. Your program buffer will be clear. If you have prepared a FIO-DEC program using the off-line flexowriter, you may read it into the EXPENSIVE TYPEWRITER text buffer by typing r. In reading the tape in, first, insert your tape with the 5-holed side toward you. The input side is the right and the tape feeds from right to left.

Turn the reader switch on at the main console by lifting it up. Typing r will read the tape into the buffer. After the tape is read in, turn the reader switch off by pressing it down and remove the tape from the reader. If a tape of your program has not been made, typing A, then your program, will put your program into the buffer. After correcting any typing mistakes using the delete, change, etc. commands, you will be ready to enter the POSSIBLE program where your program will be assembled. To enter POSSIBLE type a capital N.

When you have a program in E.T.'s buffer, you can get a punched copy by typing the command P. Your tape will automatically be punched. After punching, press the tape-feed button on the main console to give you a couple of extra fanfolds of tape, and tear off your tape. This tape can now be used on another attempt to run your program. In this way it is not necessary to retype your entire program every time you run. You should also get a typewritten copy of the machine language program by typing W. Lifting Sense Switch 1 while the buffer is being typed out by a W will stop the typing.

The E.T. has one all-purpose error signal - ?. If a ? is typed by E.T. after you have typed something, examine what you have done carefully.

Important Reminders:

- 1) The meaning of the backspace key to EXPENSIVE TYPEWRITER differs depending upon the mode of operation. If E.T. is in the control mode, the backspace key means: print out the next line (if no previous line has been mentioned, this will print out the first line). If E.T. is in the text mode, the backspace key has two meanings. If a letter of text has been typed by you a backspace erases it. If a backspace follows a carriage return typed by you, it means: return to the control mode. Do not attempt to type a command to EXPENSIVE TYPEWRITER unless it is in the control mode.
- 2) After you type in your program and leave EXPENSIVE TYPEWRITER, you will probably want to return later to make further changes to your program. When entering E.T. from DDT the second time, be sure to enter by typing 61U then 2G. A OG will erase your program entirely.
- 3) Remember every lower case command to EXPENSIVE TYPEWRITER must be followed by a carriage return before it is executed.
- 4) It is a good idea to print out the line you wish to modify using the command N1 before and after the change is made. This way you can be sure that you are changing the right line and that the change was made correctly.
- 5) After making any changes to your machine-language program with EXPENSIVE TYPEWRITER you must always reassemble your program with POSSIBLE if you want to get a new binary program. Therefore, you should leave EXPENSIVE TYPEWRITER using the N command to POSSIBLE rather than the b command to DDT.

### POSSIBLE

The POSSIBLE program when entered by N will assemble the machine-language program which is located in the buffer of EXPENSIVE TYPEWRITER. The assembled binary program is placed automatically on drum field 1, unless specified that a tape is desired.

#### Commands to POSSIBLE:

- s: begin assembly
- c: continue assembly
- a: punch symbols
- b: transfer to DDT
- k: constants area

#### Operating Procedure:

When you enter POSSIBLE with your machine-language program stored in the buffer of EXPENSIVE TYPEWRITER, typing a s will begin pass 1 of the assembly. If an error is discovered during assembly an error signal will be printed out in the format described below and the assembly process will cease.

If an error is found on pass 1, first type c to continue pass 1. Then return to EXPENSIVE TYPEWRITER by typing b then 61U then 2G, in order to correct your machine language program. After the corrections have been made, return to POSSIBLE with N and try pass 1 again. If pass 1 assembles correctly, no error message will be printed out. In this case, type s to initiate pass 2. Error in assembly during pass 2 result exactly as in pass 1 and you should proceed as described above.

If both passes assemble correctly you may, if you desire, receive a paper tape punched with your table of symbols by typing a. Also by typing k the constants storage area will be printed out. After your program is assembled by POSSIBLE, you will want to return to DDT to run it and find errors if they exist. Typing b will cause you to leave POSSIBLE and return to DDT.

Error Signals:

Upon detecting an error, POSSIBLE will print out the following:

aaa        bbbb        ccc        dddd        eee

aaa is the three letter code indicating the error. bbbb is the octal address at which the error occurred. ccc is the symbolic address at which the error occurred. dddd is the name of the last pseudo-instruction encountered. In the case of an error caused by a symbol, eee will be the symbol. Following is a list of the error indications in POSSIBLE.

<u>ERROR</u>	<u>MEANING</u>
nca	NO CONSTANTS AREA The pseudo-op <u>constants</u> is needed.
illf	ILLEGAL FORMAT
ilt	ILLEGAL TAG Tag which is not a single symbol is not equal to current location. ex. foo+10, # current location
mdt	MULTIPLY DEFINED TAG Tag consisting of a single defined symbol is not equal to current location. Symbol is not redefined.
usw	UNDEFINED SYMBOL A symbol which has not been defined in program is encountered. Symbol is given the value of zero if assembly is continued.

ERROR

cld

MEANING

CONSTANTS LOCATION DIFFERENT

The constant pseudo-op appears in different location on pass 2. No recovery can be made.

vld

VARIABLES LOCATION DIFFERENT

Same as cld but for variables. Often possible to recover by ignoring this.

lld

ILLEGAL DEFINITION

Program attempts to redefine pseudo-op or previously defined symbol. Redefines pseudo-op or symbol if assembly continued.

sce

STORAGE CAPACITY EXCEEDED

Storage of macro definitions, macro arguments, repeat ranges, numerical constants (pass 1), unique constants (pass 2), symbols, or macro names has been filled. No recovery can be made.

pce

PUSH DOWN CAPACITY EXCEEDED

Macro, repeat, or constant nesting is too deep or too complicated arithmetic statements are used. No recovery can be made.

tmc

TOO MANY CONSTANTS AND VARIABLES PSEUDO-OPS

Total number of constants and variables pseudo-ops in 208. No recovery can be made.

mdd

MULTIPLY DEFINED DIMENSION

Symbol representing first location in dimension of array is already defined. The old symbol definition is retained if assembly is continued.

tmt	TOO MANY <u>TERMINATE</u> PSEUDO-OPS There exist more <u>terminate</u> instructions than <u>define</u> instructions. The <u>terminate</u> is ignored if assembly is continued.
ids	ILLEGAL DIMENSION SIZE Specified dimension size is negative. Dimension size is set to zero if assembly continued.
ich	ILLEGAL CHARACTER Input source has an illegal flexo code or character. Number typed is the illegal character; if the number is in the 400's it is an upper case character. Continuing assembly will ignore the character.

DDT:

This program is a very valuable aid in debugging and testing your binary program. It allows you to examine the contents of locations of core, to change the contents of these locations, to run your program, and to make use of breakpoints. When in DDT, all typing will be performed in black. After a command is typed by you, DDT will type a carriage return to indicate that the command was carried out. Therefore, do not type a carriage return after a command as you would do in EXPENSIVE TYPEWRITER.

Commands to DDT

Control

2T Typed immediately after returning from POSSIBLE assembly. This places your symbols and their values into the DDT symbol table. This allows you to use symbolic addresses when communicating with DDT.

NU where N is a number. This takes the program from drum field N and places it in core memory so that it can be executed. Your binary program has been placed on drum field 1. EXPENSIVE TYPEWRITER is on absolute drum field 21. MACRO is on absolute drum field 22. (Absolute drum fields are indicated by the sign bit being on; thus, absolute field 21 is obtained by typing 61U.)

locG This will cause the computer to begin executing the program which is in core starting at location loc. loc may be either a symbolic expression or an absolute number.

For an example of the above commands:

2T  
1U  
begG

will enter your symbol table from POSSIBLE into DDT's symbol table from POSSIBLE, bring your program from drum field 1 into core, and begin executing your program at location beg.

Inspecting and Modifying:

Examine a Register

adr/: Types out the contents of register adr.  
"adr" may be a symbolic address or an  
octal number.

Examples: beg+2/ types out the contents  
of the register 2 positions  
past beg. 55/ types out  
the contents of register 55.

Change a Register

adr/ exp: Changes the contents of register  
adr to exp.

Example: k/ 4 132 examines  
the contents of the register k and  
finds it contains the number 4. This  
is changed to 13.

Set the Mode

S Sets DDT to type out all words as  
symbolic instructions.

C Sets DDT to type out all words as  
octal number.

Examples: S  
20/ and k  
C  
20/ 020003 (if k = 3)

U

Converts all numeric printouts to decimal .

H

Resets DDT so that all numeric printout are in the normal octal mode again.

Examples:

C

20/

020003

UJ

20/

8195

H

20/

020003

#### Convert an Expression

exp =

Types out the value of the expression exp as an octal number.

num →

Types out the octal number num as an instruction.

char ~

Types out the character (s) char as concise code.

Examples: 20/

20003 → and K  
(if k = 3)

beg =

20 (if beg is the symbol for location 20)

a40~

610420

### Examining Sequences of Register

adr/ ... ↑      Opens the register preceding adr and types its contents.

Example: tab/    13    ↑  
tab-1/   21    ↑  
tab-2/    0

Register tab contained 13, tab-1 contained 21, and tab-2 contained 0.

adr/ ... (bs)      Opens the contents of the register next in sequence to adr and types out.

(bs) means back-

space      Example: tab/    13    (bs)  
tab+1    61

Register tab+1 contains 61

adr/ ... (tab)      Types out the contents of the register addresses by the contents of register adr.

Example: beg/    lac 13 tab 44

Register 13 contains 44

### Breakpoint Commands

It is tempting to load a newly coded program into the computer, let it run to completion, and check the final answer for correctness. Because of the high probability of one or more errors in even short programs and the cumulative effect of these errors, this approach is practically worthless (except for very short programs).

A more satisfactory approach is to run only a small portion of the program at a time and check intermediate results. This allows one to catch errors before they have a chance to make the whole operation unintelligible. The breakpoint feature of DDT permits one to take this approach and check out one section of the program at a time.

A breakpoint is a place in the user's program where computation is interrupted and control is transferred to DDT. This is accomplished by removing and saving the instruction at the breakpoint and inserting in its place a jump instruction to DDT.

When the program reaches this jump, the status of the machine (AC, IO, overflow indicator, etc.) is saved and the breakpoint location and contents of that location are typed out.

At this point one may examine the intermediate results by use of any of the inspection and modification commands listed above. If they are satisfactory, one may proceed with the next section of the program. If they are in error, one may try to correct the errors and re-run the section. The DDT breakpoint commands are as follows:

locB Prepares DDT to insert a breakpoint at location loc. The actual insertion occurs when a G or X command is given.

B Removes the previous breakpoints.

P Given after a program has been interrupted by a breakpoint. The instruction removed for the breakpoint is executed and control is returned to the user's program.

insX Causes instruction ins to be executed.

Examples:

```
lupB  
begG  
lup)      add sym  
k/          4  
B  
P
```

This example places a breakpoint at location lup and transfers control to beg. The content of register lup is add sym and location k contains 4 at the break. The breakpoint is removed and the program is allowed to proceed.

dzm conX deposits +0 into register con